

Name of Student: Pushkar Sane		
Roll Number: 45		Lab Assignment Number: 4
Title of Lab Assignment: To implement programs on Python Functions and Modules.		
DOP: 07-10-2023		DOS: 09-10-2023
CO Mapped: CO1	PO Mapped: PO3, PO5, PSO1, PSO2	Signature:

Practical No. 4

Aim: To implement programs on Python Functions and Modules.

Description:

- **Function In Python:**

- a. A function in Python is a named block of code that performs a specific task or set of tasks.
- b. Functions are essential for organizing and structuring your code, promoting code reuse, and making your code more readable and maintainable. They encapsulate a series of instructions, allowing you to call and reuse them throughout your program.

- **Function Syntax:**

Here's the general syntax of defining a function in Python:

```
def function_name(parameter1, parameter2, ...):  
    # Function body - code to perform the task  
    # You can use parameters in the function body  
    result = parameter1 + parameter2  
    return result # Optional, specifies the value to be returned
```

- a. **def:** This keyword is used to define a function in Python.
- b. **function_name:** Choose a descriptive name for your function that reflects its purpose. Function names should follow naming conventions (e.g., use lowercase letters and underscores).
- c. **parameter1, parameter2, ... :** These are optional input parameters or arguments that the function accepts. Parameters allow you to pass data into the function for processing.
- d. **':' :** A colon is used to denote the beginning of the function's body.
- e. **Function body:** The function body consists of the actual code that performs the desired task. It can include calculations, loops, conditionals, and other statements.

- f. **return:** This keyword, followed by an expression, is used to specify the value that the function should return as its result. Not all functions need to return a value; it's optional.

Example: Adding Two Numbers:

```
def add_numbers(num1, num2):  
    result = num1 + num2  
    return result  
  
# Call the function with arguments 5 and 3  
sum_result = add_numbers(5, 3)  
print(sum_result) # Output: 8  
  
# Call the function with different arguments  
result2 = add_numbers(10, 20)  
print(result2) # Output: 30
```

- **Recursion:**

Recursion is a programming technique in which a function calls itself to solve a problem. In Python, you can implement recursive functions to break down complex problems into simpler, self-similar sub problems. To successfully use recursion, you need to define a base case (the terminating condition) and one or more recursive cases (the cases where the function calls itself).

Here's a detailed explanation of recursion with an example:

Factorial Calculation Using Recursion:

```
def factorial(n):  
    # Base case: When n is 0 or 1, the factorial is 1.  
    if n == 0 or n == 1:  
        return 1  
  
    # Recursive case: Calculate factorial by calling the function recursively.  
    else:  
        return n * factorial(n - 1)
```

In this recursive function:

- The base case is when `n` is either `0` or `1`. In these cases, we return `1` because `0!` and `1!` are both equal to `1`.
- The recursive case calculates the factorial of `n` by calling the `factorial` function recursively with the argument `n - 1`. This reduces the problem to a smaller subproblem.

- **Lambda Function**

In Python, a lambda function is a small, anonymous, and inline function defined using the `lambda` keyword. Lambda functions are also known as anonymous functions because they don't have a name. They are typically used for short, simple operations where defining a full function using the `def` keyword would be overly verbose.

The basic syntax of a lambda function is as follows:

`lambda arguments: expression`

- lambda:** The `lambda` keyword is used to indicate the creation of a lambda function.
- arguments:** These are the input parameters to the lambda function, separated by commas. Lambda functions can take any number of arguments but are often used with one or two.
- expression:** This is a single expression that the lambda function evaluates and returns as its result.

Here's an example to illustrate how lambda functions work:

Regular function to add two numbers

```
def add(x, y):
```

```
    return x + y
```

Equivalent lambda function

```
add_lambda = lambda x, y: x + y
```

Using both functions to add numbers

```
result1 = add(5, 3)
```

```
result2 = add_lambda(5, 3)
```

```
print("Using the regular function:", result1) # Output: Using the regular function: 8
```

```
print("Using the lambda function:", result2)
```

- **Module:**

In Python, a module is a file containing Python code, including variables, functions, and classes, that can be imported and used in other Python scripts or programs. Modules allow you to organize your code into reusable and manageable units, making your code more modular and maintainable. A file in Python can serve as a module if it contains Python code. To create a module, you typically save your Python code in a `.py` file with the same name as the module you want to create. For example, if you want to create a module named `my_module`, you could create a file named `my_module.py`.

Here's a simple example of a Python module:

my_module.py

```
# This is a Python module named 'my_module'
```

```
def greet(name):
```

```
    return f"Hello, {name}!"
```

```
def add(a, b):
```

```
    return a + b
```

```
pi = 3.14159265359
```

Now, you can use this module in another Python script by importing it using the `import` statement:

main.py

```
# Importing the 'my_module' module
```

```
import my_module
```

```
# Using functions and variables from the module
```

```
print(my_module.greet("Alice"))    # Output: Hello, Alice!
```

```
print(my_module.add(5, 3))         # Output: 8
```

```
print(my_module.pi)               # Output: 3.14159265359
```

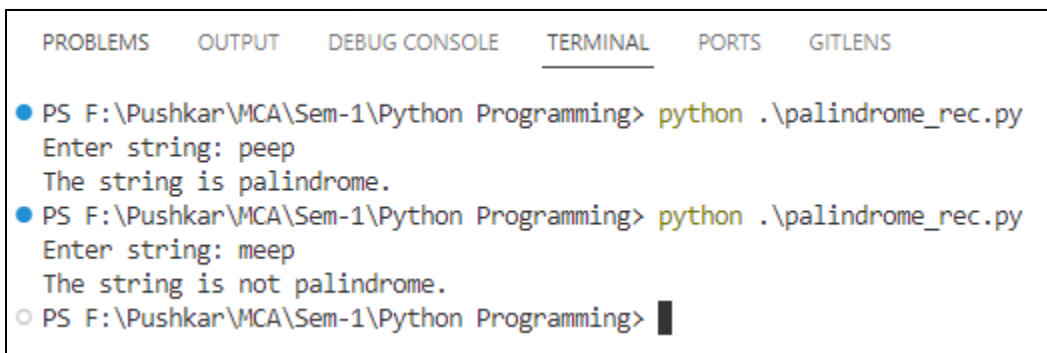
1. To check whether a string is palindrome or not using function recursion.**Code:**

```
# Define a function
def isPalindrome(string):
    if len(string) < 1:
        return True
    else:
        if string[0] == string[-1]:
            return isPalindrome(string[1:-1])
        else:
            return False
#Enter input string
str1 = input("Enter string:")

if(isPalindrome(str1)==True):
    print("The string is palindrome.")
else:
    print("The string is not palindrome.")
```

Conclusion:

Here, we reverse a string using the recursion method which is a process where the function calls itself. Then we check if the reversed string matches with the original string and demonstrate the program for checking if a string is palindrome or not using recursion.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\palindrome_rec.py
Enter string: peep
The string is palindrome.
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\palindrome_rec.py
Enter string: meep
The string is not palindrome.
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

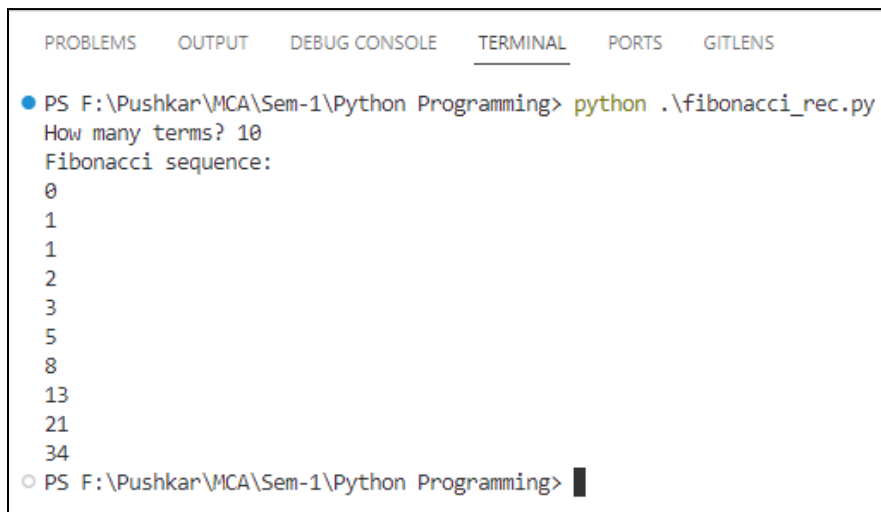
2. To find Fibonacci series using recursion.**Code:**

```
def recur_fibo(n):
    if n <= 1:
        return n
    else:
        return(recur_fibo(n-1) + recur_fibo(n-2))

# take input from the user
nterms = int(input("How many terms? "))
# check if the number of terms is valid
if nterms <= 0:
    print("Please enter a positive integer")
else:
    print("Fibonacci sequence:")
    for i in range(nterms):
        print(recur_fibo(i))
```

Conclusion:

In this program, we store the number of terms to be displayed in nterms. A recursive function recur_fibo() is used to calculate the nth term of the sequence. We use a for loop to iterate and calculate each term recursively.

Output:

The screenshot shows a terminal window with the following content:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\fibonacci_rec.py
How many terms? 10
Fibonacci sequence:
0
1
1
2
3
5
8
13
21
34
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

3. To find the binary equivalent of a number using recursion.**Code:**

```
# Decimal to binary conversion using recursion
def find( decimal_number ):
    if decimal_number == 0:
        return 0
    else:
        return (decimal_number % 2 + 10 *
                find(int(decimal_number // 2)))

# Driver Code
decimal_number = int(input("Enter a decimal number: "))
print(find(decimal_number))
```

Conclusion:

Demonstrated the program for finding the binary equivalent or converting a decimal number to binary number using recursion.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\dectobi_rec.py
Enter a decimal number: 100
Binary equivalent is 1100100
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```


4. To use lambda function on list to generate filtered list, mapped list and reduced list.

Code:

```
from functools import reduce

# Sample list
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Filter using a lambda function
filtered_list = list(filter(lambda x: x % 2 == 0, my_list))

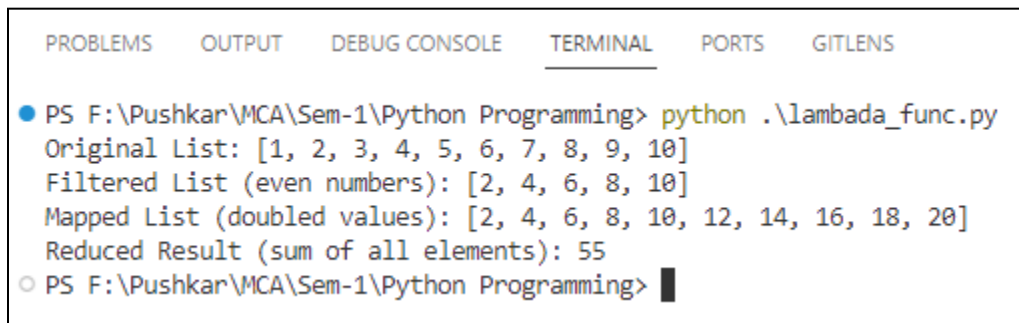
# Map using a lambda function
mapped_list = list(map(lambda x: x * 2, my_list))

# Reduce using a lambda function
reduced_result = reduce(lambda x, y: x + y, my_list)

# Print the results
print("Original List:", my_list)
print("Filtered List (even numbers):", filtered_list)
print("Mapped List (doubled values):", mapped_list)
print("Reduced Result (sum of all elements):", reduced_result)
```

Conclusion

In this Python program, we demonstrated how to use lambda functions in conjunction with the filter(), map(), and reduce() functions to perform different operations on a given list.

Output:

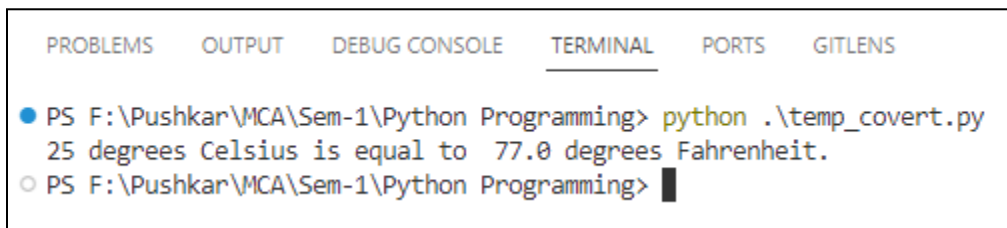
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\lambda_func.py
Original List: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Filtered List (even numbers): [2, 4, 6, 8, 10]
Mapped List (doubled values): [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
Reduced Result (sum of all elements): 55
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

5. Convert the temperature in Celsius to Fahrenheit in a list using an anonymous function.**Code:**

```
# Convert Celsius to Fahrenheit using a lambda function
celsius_to_fahrenheit = lambda c: (c * 9/5) + 32
# Temperature in Celsius
celsius_temperature = 25
# Use the lambda function to convert
fahrenheit_temperature = celsius_to_fahrenheit(celsius_temperature)
# Print the result
print(celsius_temperature, "degrees Celsius is equal to ", fahrenheit_temperature ,
      "degrees Fahrenheit.")
```

Conclusion:

This program demonstrates how to perform a straightforward temperature conversion on a list using an anonymous lambda function and then display the results. It's a concise way to achieve the desired conversion without the need for additional functions or complex string formatting.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> python .\temp_covert.py
  25 degrees Celsius is equal to 77.0 degrees Fahrenheit.
○ PS F:\Pushkar\MCA\Sem-1\Python Programming> █
```

6. To create modules in python and access functions of the module by importing it to another file/module. (Calculator program)

Code:

Calculator.py

```
# calculator.py
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
    if y == 0:
        return "Error: Division by zero"
    return x / y
```

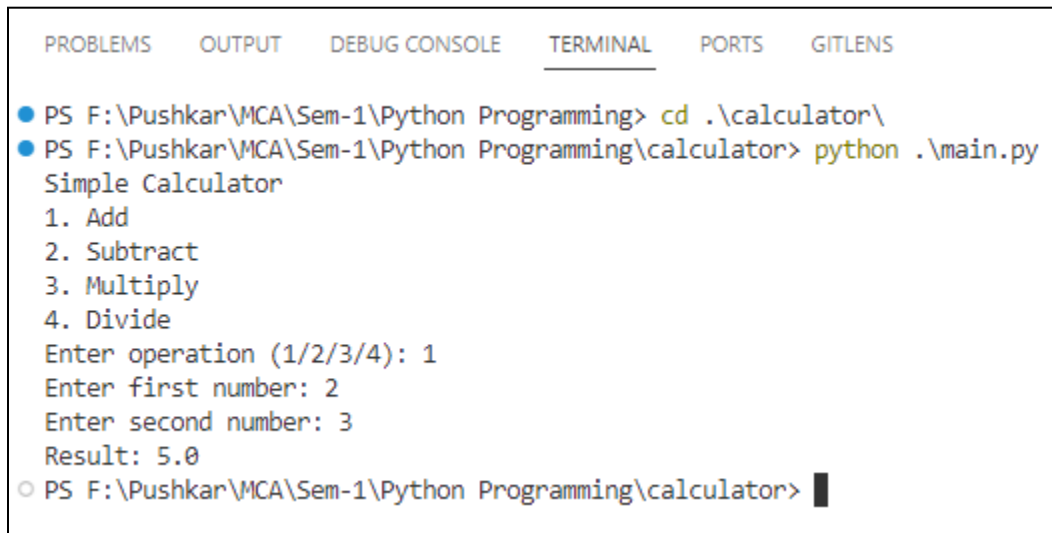
Main.py

```
# main.py
import calculator
def main():
    print("Simple Calculator")
    print("1. Add")
    print("2. Subtract")
    print("3. Multiply")
    print("4. Divide")
    choice = input("Enter operation (1/2/3/4): ")
    num1 = float(input("Enter first number: "))
    num2 = float(input("Enter second number: "))
    if choice == '1':
        result = calculator.add(num1, num2)
        print(f"Result: {result}")
    elif choice == '2':
        result = calculator.subtract(num1, num2)
        print(f"Result: {result}")
```

```
elif choice == '3':
    result = calculator.multiply(num1, num2)
    print(f"Result: {result}")
elif choice == '4':
    result = calculator.divide(num1, num2)
    print(f"Result: {result}")
else:
    print("Invalid choice")
if __name__ == "__main__":
    main()
```

Conclusion:

Here, we've demonstrated the creation of a modular calculator program using two separate files. The calculator.py module encapsulates core arithmetic functions, while the main.py script serves as a user interface, importing and utilizing these functions. This modular approach enhances code organization, reusability, and readability, making it easier to maintain and expand the calculator program.

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● PS F:\Pushkar\MCA\Sem-1\Python Programming> cd .\calculator\
● PS F:\Pushkar\MCA\Sem-1\Python Programming\calculator> python .\main.py
Simple Calculator
1. Add
2. Subtract
3. Multiply
4. Divide
Enter operation (1/2/3/4): 1
Enter first number: 2
Enter second number: 3
Result: 5.0
○ PS F:\Pushkar\MCA\Sem-1\Python Programming\calculator> █
```